# .Net Framework:-

## Introduction

.Net Framework is the first step to enter in to the .Net world. A framework can be defined as building blocks. Similarly .Net Framework can be defined as Building blocks of any .Net application. The .Net framework contains set of assemblies (in .Net we call all the DLLs as Assemblies) to support different kind of .Net applications (may be windows based, web based and other applications). Depending upon the kind of application we would like to develop we can use the corresponding Assemblies available in the .Net Framework. .Net Framework also contains some Assemblies to execute the .Net application. So *.Net Framework is the collection of Assemblies* to support the .Net application development and .Net application execution.

## The Word .Net:-

We know that .Net Framework is a collection of Assemblies. Now let us know what does the word .Net mean? There is no reason given by Microsoft for the word ".Net", but you can see some definitions over the web saying that it is "Network of all technologies", "Next Generation Technology" and so.

## Why do you need .Net?

We have so many technologies available over the world. Now why do we need one more technology called .Net? Why should I go for .Net? The answer for this question can be

**1. Automatic Memory Management**

**2. Web Services**

**3. Easy Deployment**

**4. Interoperability**

**5. Net Language Independent**

Let's look on each briefly,

### Automatic Memory Management:-

The most expensive problem the current VC++ programmers are facing would be 'memory leak'. To be clear, every new operator we use in our program should be matched with a delete operator. i.e. if we allocate memory for an object using new keyword then it should be deallocated using the delete operator. If we forget to apply delete operator then the memory allocated using the new operator cannot be used for other purposes. This is called memory leak. In .net there is no place for the term memory leak, because we don't have to use delete operator

on any object. We can use the new keyword, and forget about delete. .Net will automatically delete the object when that particular object is no longer used or referenced.

### Web Services:-

Web Services are standardized way to communicate between two entities/processes over web. This uses XML(eXtensible Markup Language), WSDL (Web Service Description Language), SOAP (Simple Object Access Protocol), and UDDI (Universal Description, Discovery and Integration). We can think of this web services as similar to windows services but this web services can also be accessed from any where over web. .Net has full support to the web services comparing other technologies.

### Easy Deployment:-

In case of .Net applications, the deployment is just a copy and paste. Because all the .Net assemblies are 'self describing'. We don't need to depend on any header file or registry (like COM components) or others. Just by copying the files and pasting it in a different machine will complete the deployment. (However any shared assemblies should be installed into a special folder called 'assembly').

### Interoperability:-

Interoperability can be defined as the ability of a hardware/software to share data between two applications on different/same machine. Consider we have two applications one is developed in VC++ and other one is developed in VB. It is not possible to share the data (with out any special mechanism) from one application to another application as the data type used in one language cannot be understood by other application.

Most of the applications running now a day were developed using C, C++, COM etc. Moving towards .Net technology, it is not possible to recreate all the application developed in other technologies so far. For that purpose .Net has provided a full support for .Net applications to interact with other applications and vice versa. Any COM component or Win32 application can be communicated easily with .Net application.

### .Net Language Independent:-

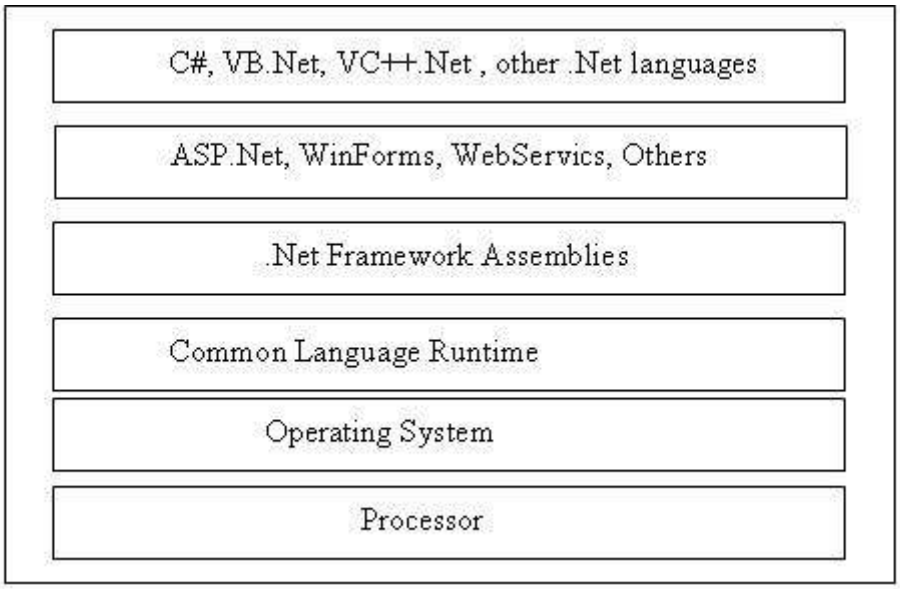
Consider you are creating a DLL using VC++. Now what if you want use this DLL in VB application, we cannot use the DLL in VB, i.e. cross language interoperability is not supported. But in case of .Net languages you can create a DLL using C#.Net and use that DLL in VB.Net application. Similarly you can cross with any .Net languages. Note that only .Net languages can be crossed. VC++.Net and VC++ are not same (we will see the difference later).

## What is .Net?

So what is .Net? It is not a Language. It is not an operating system. It is a platform to execute .Net applications.

## .Net Architecture:-

We saw that .Net is platform to execute the .Net applications. Let's now look at the architecture of the .Net




The above diagram summarizes the architecture; let's look at the each item from the bottom,

***Processor***: Needless to say all the applications should run in processor.

***Operating System***: similarly all the application can get the processor time only through the operating system.

***Common Language Runtime***: It is usually called as CLR, which is a new layer added for .Net architecture. This new layer differentiates .Net applications from other applications. Normal windows applications run directly under the operating system, but any .Net application cannot run directly under the operating system, all the .Net applications require CLR to be there to get executed.

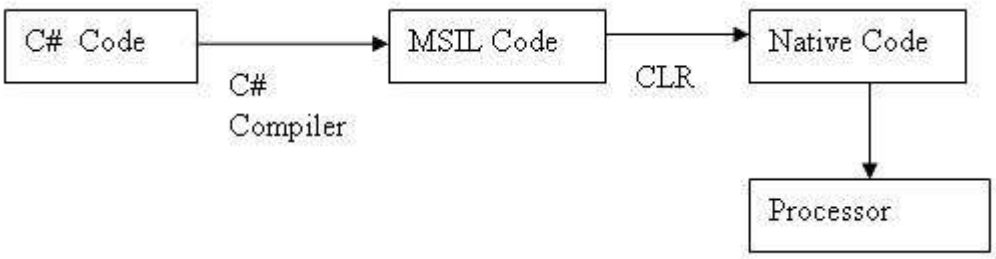
***.Net Framework Assemblies***: These are the set of assemblies can be used by our application. .Net has provided a lot of inbuilt classes and methods which can be used by our application. These classes and the methods are packed in different assemblies. All these assemblies are collectively called .Net framework assemblies.

***ASP.Net, WinForms, WebServics, Others***: These are kind of application we usually develop. ASP.Net refers to the web based applications, Winforms refers to the Windows based application, WebServices as seen earlier. And there are other kinds of application we can develop as per the requirement. Each kind of application uses respective .Net framework assemblies. Not all applications use all assemblies. Each application uses related or corresponding assemblies from the .Net framework.

***.Net Languages***: These are the programming languages we use to develop our application. We can choose between any languages as per our comfort. All the languages have the full access to the .Net framework. (However some .Net features are specially designed for C#).

## .Net application Execution Steps:-

We saw that .Net architecture contains a new layer called CLR, without this layer any .Net application can not be executed. Now let us see the steps involved in a program execution.




Before starting with .Net application execution let's recollect how a normal C program works.

We write our C code in plain ASCII code, the C compiler converts the ASCII code to binary code (.exe), this binary code will be executed by the operating system.

But in case of .Net, the compiler will not generate binary code from the ASCII code, instead the compliers will generate only the MSIL (MicroSoft Intermediate Language) from the ASCII code. This MSIL will then be converted to binary or native code by the CLR depending upon the target OS. That is why we need CLR to execute any .Net application as the processor cannot understand the MSIL code they understand only the binary code.

Why? Why do we want to convert the ASCII to MSIL first then to Native? What is the advantage do we get?

The reason why do we go for the MSIL step is that, we are getting most of the benefits of .Net through this MSIL, for e.g. resource management, object lifetime management, reflection, type safe, some debugging support, etc. To avail these benefits we go for the MSIL step. And these benefits make the developer's life very easy. Finally we, the developers are rewarded with this MSIL.

## Is .Net Platform Independent?

Before answering this question let's recollect what is platform independence. It can be defined as 'compile the code in one platform and run it in any platform'. To be clear the compiler should not generate any platform specific code as compiler output.

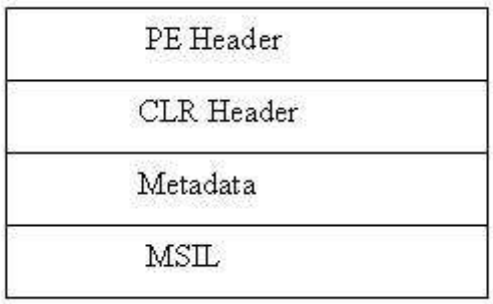
How java is platform independent? Java compiler generates byte code as complier output. This byte code is platform independent. And java has JVM for different platform, the byte code can run on any JVM i.e. any platform. Thus the generated byte code is platform independent.

Coming back to .Net, as we saw, our .Net compliers generates the MSIL code and this MSIL will run on CLR. This MSIL is not platform specific i.e. it can run on any platform if CLR for the platform is available. But Microsoft has not provided CLR other than Windows platform. So .Net technology is platform independent but we don't have CLR for other platforms from Microsoft.

## Identifying a .Net application:-

We saw that we need CLR to run any .Net application. Consider you have two EXEs. One is .Net exe and other one is normal windows application exe. Both are located in your C drive. When you double click on an exe the OS (Windows) will handle the click event and will run the exe in the processor. But incase of .Net exe the CLR has to be loaded to execute the .Net application. So how does the OS know to load CLR incase of .Net application? To answer this question let's understand the parts of a .Net assembly/exe.

Any .Net assembly/exe has the following four sections.




***PE Header*** is a normal --Common Object File Format -- COFF header used by the operating system. When we double click on any exe the instruction inside this section will be executed by the OS. In our case, for .Net applications this section will redirect the OS to the CLR header section.

***CLR Header***, this is a different section which we have only in .Net assemblies. This section contains some instructions to load the CLR. This part of header section is responsible to load the CLR (if not been loaded already) when we try to execute a .Net application. Till now the control is in the hands of OS, after the CLR is loaded the CLR will take the control of the application from the OS and the CLR will execute the .Net application. Now the CLR will execute the application from the Main method available in the MSIL section.

***Metadata***, initially we saw that all the .Net assemblies are 'self describing'. This metadata section is responsible self description. This section will contains all the information about the assembly like list of namespaces, classes it has and methods in each class, parameters, etc.

***MSIL***, implementation of all the methods, classes, etc. available in the assembly will be there in this section in MSIL format. As we saw this is not in binary format. MSIL is like another assembly language.

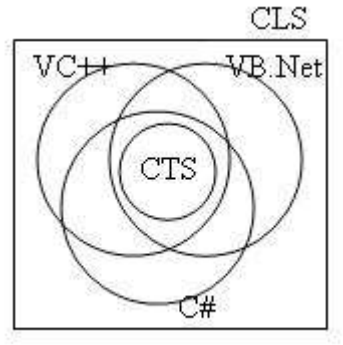
## Does Language Matter?

We have VB.Net, C#, VC++.Net, J#, etc. languages. Does choosing a language for an application affects the performance or usability of framework? The answer is NO. All the compliers generate the MSIL code. Microsoft has provided these many languages to support different syntax. If you are familiar with VB syntax then you can go for the VB.Net language, if you are familiar with VC++ syntax you can go for the VC++ language. Choosing a language does not affect the usage of .Net framework. Al the framework libraries are available for all the languages. And as all the

compliers generate the MSIL it doesn't have any impact on the runtime too. If you are new to any syntax it is good to use C# as it has got more features.

## CLS/CTS:-

As we saw we have many .Net languages but all the languages produce the MSIL as output. This MSIL is common. So the target types are shared by all the languages. These target type are the types that is defined in the MISL. And all the languages have to obey some rules for generating the MSIL like syntax and types. The rules that should be followed by any .Net languages are called CLS or Common Language Specification. And there are some set of types in the MSIL that should be supported by any language as a minimum requirement. These set of types are called CTS or Common Type System. There are some types outside the CTS, but these type are optional may be supported by a language. Each language will have its own boundary like the following diagram.




In the above diagram we could see that all the languages supports the CTS minimally and some extra types. And diagram also shows all the languages obey the rules defined in the CLS. We achieve the interoperability through this CTS. Since all the .Net languages are expected to support CTS it is guaranteed that types inside this CTS are fully interoperable.

Having said about CTS/CLS it is also possible to write your own .Net language which will have your own syntax. But while developing the complier you should keep in mind that you support all the CTS Types and obey the rules specified in CLS.